

Ocpn_Draw ODAPI

Updated ODAPI for faster access OCPN_Draw capabilities

Version 1.1

This update introduces the ODAPI which allows other plugins and OCPN to gain direct access to the JSON API functions without the overhead of JSON processing. You will need to include ODAPI.h in your code and then use a JSON message to get the addresses of all the API's.

Example to get the api's:

```

wxJSONValue jMsg;
wxJSONWriter writer;
wxString    MsgString;
jMsg[wxT("Source")] = wxT("WATCHDOG_PI");
jMsg[wxT("Type")] = wxT("Request");
jMsg[wxT("Msg")] = wxS("GetAPIAddresses");
jMsg[wxT("MsgId")] = wxS("GetAPIAddresses");
writer.Write( jMsg, MsgString );
SendPluginMessage( wxS("OCPN_DRAW_PI"), MsgString );
if(g_ReceivedODAPIJSONMsg[wxT("MsgId")].AsString() == wxS("GetAPIAddresses")) {
    wxString sptr =
g_ReceivedODAPIJSONMsg[_T("OD_FindPointInAnyBoundary")].AsString();
    if(sptr != _T("null")) {
        sscanf(sptr.To8BitData().data(), "%p",
&pOD_FindPointInAnyBoundary);
        l_bUseODAPI_FPIAB = true;
    }
    sptr =
g_ReceivedODAPIJSONMsg[_T("OD_FindClosestBoundaryLineCrossing")].AsString();
    if(sptr != _T("null")) {
        sscanf(sptr.To8BitData().data(), "%p",
&pODFindClosestBoundaryLineCrossing);
        l_bUseODAPI_FCBLC = true;
    }
}

```

Example calling api's:

```

FindClosestBoundaryLineCrossing_t *pFCPIAB = new
FindClosestBoundaryLineCrossing_t;
pFCPIAB->dStartLat = lastfix.Lat;
pFCPIAB->dStartLon = lastfix.Lon;

```

```
pFCPIAB->dEndLat = lat;  
pFCPIAB->dEndLon = lon;  
(*pODFindClosestBoundaryLineCrossing)(pFCPIAB)
```

An example of the use of the API can be found in the WatchDog plugin. You will need to get at least 1.9.048 of the WatchDog plugin to see examples of its use.

The ODAPI.h file contains typedef statements that make it easier to use the API. All the API's are defined here so you use the typedef to define a variable to hold the address of the correct API and then use this to execute the API.

Additional Notes about ODAPI

The original ODAPI only had four functions which were read only:

- OD_FindPathByGUID - Find a path of any type by its GUID
- OD_FindPointInAnyBoundary - Find if a Lat/Lon is inside any boundary
- OD_FindClosestBoundaryLineCrossing - Find if a line crosses a boundary line and will return the closest line crossing
- OD_FindFirstBoundaryLineCrossing - Find if a line crosses a boundary and return as soon as a crossing is detected

These were initially used by the WatchDog and WeatherRouting plugins.

With OD 1.5 (beta) and OD 1.6 there are new API's available to allow creation of objects. These are:

- OD_CreateBoundary - Create a Boundary
- OD_CreateBoundaryPoint - Create an isolated Boundary Point
- OD_CreateTextPoint - Create a Text Point
- OD_DeleteTextPoint - Delete a Text Point
- OD_AddPointIcon - Add an OD Point icon that can be used
- OD_DeletePointIcon - Delete a previously added OD Point icon

This now allows use of some of the OD drawing tools by other plugins. An updated version of Squiddio_pi is available, also beta, which can use either OCPN Waypoints or OD Text Points for the display of information. The change has been made to the method of displaying the information but not to the functionality of Squiddio.

To help with understanding of how to use the new API's and for testing purposes a new plugin has been created, testplugin_pi (https://github.com/jongough/testplugin_pi). This plugin can use either the ODAPI binary interface or JSON text form to drive OD. The names of the API's should be self explanatory as to what they do and reference to the testplugin should help with understanding.

Additionally Jon Gough writes:

https://github.com/seandepagnier/weather_routing_pi/issues/191#issuecomment-385209963

There are two methods of interacting with OD programatically, OCPN messaging using JSON messages and the ODAPI. Basically the OCPN messaging is synchronous and broadcast, so every plugin gets the message being sent and the originator is blocked until the message process completes. Also, the message is in JSON format which requires decoding, so every plugin that is registered for messaging has to decode the message to see if it is interested in it. The ODAPI uses the messaging process to give out the real address of the API, but once this is done it is possible to call the available OD functions directly with a C structure.

So the OCPN JSON message is useful for letting multiple plugins know what is going on and for low message rate processes. For targeted, high rate, message/communications it is not really usable in its current format (Dave said he was going to look into it for OCPN V5). For Watchdog and Weather Routing the message rate can be high and it is only for OD, no one else should be interested in the messages, so the ODAPI was created to avoid the JSON overhead and to allow plugins to communicate directly with OD (in fact execute OD code as it was part of their plugin).

Jon advises

Basically the ODAPI is defined in ODAPI.h which should be used by any plugin wishing to use the binary interaction method, there is also the ODJSONSchemas.h file which defines the JSON version of the API in a schema format. There is as yet no globally usable parser for this version of the schema (still waiting for another open source development to catch up with the JSON standards).

From:
<https://opencpn.org/wiki/dokuwiki/> - **OpenCPN Manuals**

Permanent link:
https://opencpn.org/wiki/dokuwiki/doku.php?id=opencpn:developer_manual:plugins:plugin_api:ocpn_draw_odapi

Last update: **2019/05/20 08:30**

